



Introduzindo os React Hooks: o que são e como usar

Os Hooks foram introduzidos na versão 16.8 do React. Resumidamente falando, são funções que permitem a ligação entre os componentes funcionais de uma aplicação aos recursos de state e ciclo de vida do React. Esses recursos foram inseridos na biblioteca tendo como objetivo simplificar uma série de tarefas que, até então, eram possíveis apenas com muitas linhas de códigos.

Um ponto importante em relação aos Hooks é que há vários já disponíveis para utilizarmos e, além disso, também temos a possibilidade de criarmos os nossos próprios hooks. Na lista abaixo, podemos ver a relação das funções pré-existentes:

- useState;
- useEffect;
- useContext.

Além desses, chamados de Básicos, há ainda outros, os Adicionais:

- useReducer;
- useCallback;
- useMemo;
- useRef;



- useImperativeHandle;



- useLayoutEffect;
- useDebugValue.

Como de praxe em tudo relacionado à programação, nada melhor para entender algo novo do que vermos código (e praticarmos, é claro). A seguir, veremos alguns exemplos de Hooks – tirados da documentação oficial do React. Vamos nos aprofundar, de maneira especial, em dois Hooks: o useState e o useEffect.

```
import React, { useState } from 'react';

function Example() {
  // Declara uma nova variável de state, que chamaremos de "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Clique aqui
      </button>
    </div>
  );
}
```

Figura 1: Hook useState

```
function ExampleWithManyStates() {
  // Declara várias variáveis de state!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
  // ...
}
```

Figura 2: Múltiplos useState

```
function Example() {
  const locale = useContext(LocaleContext);
  const theme = useContext(ThemeContext);
  // ...
}
```

Figura 3: Hook useContext

Após nosso (provável) primeiro contato com os Hooks, cabe ressaltar alguns detalhes vistos nas figuras acima. O primeiro deles é que os hooks useState e useContext possuem formas bastante similares de declaração, embora tenham funções bastantes distintas (enquanto o useState manipula os estados que criamos em um componente, o useContext lida com o Contexto – que pode ser considerado um armazenamento global em uma aplicação). Em ambos, declaramos uma constante que recebe a chamada do Hook juntamente com algum parâmetro. Dentre os Hooks pré-existentes, normalmente, utilizaremos com mais frequência os básicos, dos quais ainda não vimos exemplo apenas do useEffect, o que faremos mais adiante.

A partir de agora, olharemos um pouco mais a fundo dois hooks, o useState e o useEffect. Começando pelo useState, que é um recurso que nos permite manipular o state ou states em nossos componentes. Lembrando que um state, no React, é uma variável responsável pelo armazenamento de estados dentro de um componente. Nesse sentido, o uso de state permite ao componente alterar seu comportamento e resultado a partir de ações dos usuários e outros eventos. Se olharmos a Figura 1, acima, veremos que foi declarado um state chamado "count" e que esse state é alterado a partir da ação do usuário de clicar no botão declarado como retorno do componente. Tal incremento é feito através do método setCount. A respeito desse método, perceba também que

ele é declarado juntamente com o state. Nesse ponto, temos o formato completo de declaração do Hook useState: o state e[illegible], e o método de manipulação, cujo nome é composto pelo nome do state precedido pelo termo “set”. Reveja abaixo a declaração do state count:

```
...
const [count, setCount] = useState(0);
...
```

Agora, imagine que você deseja criar um state chamado temperatura, responsável por manter, em uma aplicação, o valor da temperatura ambiente. Além disso, imagine também que a temperatura poderá ser alterada com o passar do tempo, sendo atualizada por algum evento, como a consulta de uma API externa, por exemplo. Nesse caso, nosso state e useState seriam declarados dessa forma:

```
...
const [temperatura, setTemperatura] = useState(0);
...
```

Nos dois últimos exemplos, perceba que o useState foi declarado recebendo um argumento – o número zero. Esse parâmetro, passado para o Hook em sua declaração, é o seu valor inicial (conhecido como initial state) e está disponível já na renderização inicial do

componente. Ao longo do ciclo de vida do componente, o valor inicial (que, inclusive, não é obrigatório, ou seja, podemos declarar um state que não tenha valor inicial ou usar outros tipos de dados na sua inicialização, incluindo arrays e também objetos) pode ser modificado com o método "set". Voltando à nossa constante temperatura, veja abaixo como podemos setar um novo valor para ela:

```
...
setTemperatura(20);
...
```

A partir desse ponto, a temperatura passaria a ter o valor "20". Com isso, nas renderizações seguintes, o valor do estado será sempre o mais recente (o que foi definido por último).

Por último, sobre o hook useState – mais precisamente sobre o método setState, é importante que você saiba que é possível combinar o estado anterior com um novo estado (realizar um "append"). Tal recurso é muito útil quando armazenamos arrays ou objetos em um state e precisamos acrescentar novos itens aos mesmos. Para isso, utilize a sintaxe abaixo:

```
...
const [temperaturas, setTemperaturas] = useState({})
setTemperaturas(prevState => {
        return {...prevState, ...updatedValues}
});;
...
```

Após vermos mais a fundo o hook useState, veremos, a partir de agora, sobre outro Hook bastante útil, o useEffect – ou Hook de efeito. Esse Hook permite que sejam aplicados efeitos colaterais ou apenas efeitos, em componentes funcionais. Vamos a um primeiro exemplo que nos ajudará em sua compreensão:

```
import React, { useState, useEffect } from 'react';

function Exemplo() {
  const [count, setCount] = useState(0);

  // Similar ao componentDidMount e componentDidUpdate:
  useEffect(() => {
    // Atualiza o título do documento usando a API do browser
    document.title = `Você clicou ${count} vezes`;
  });

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Clique aqui
      </button>
    </div>
  );
}
```

Figura 4: Hook de Efeito

Nesse exemplo, o useEffect é executado a cada evento "onClick". Repare que em seu escopo estamos atualizando dinamicamente o título da página web a cada clique do usuário no botão que altera o state count. Esses cliques re-renderizam a página, fazendo com que o useEffect seja executado.

Recomendo que você, por curiosidade, crie uma nova aplicação react e modifique o código inicial do App.js pelo código abaixo.



```
import React, { useState, useEffect } from 'react';
function App() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    // Atualiza o título do documento usando a API do browser
    document.title = `Você clicou ${count} vezes`;
    console.log('Componente re-renderizado');
  });

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Clique aqui
      </button>
```

App.js: Utilizando o useEffect

Antes de clicar no botão, após o navegador iniciar a renderização da aplicação, abra também o seu inspecionador de elementos (no navegador) e vá para a aba “console”. Agora clique no botão. Repare que o número de vezes que o botão foi clicado vai sendo incrementado, na página e também em seu título. Além disso, repare no console que a mensagem inserida na declaração do useEffect (com o “console.log”) também vai sendo incrementada. Isso prova o que foi dito anteriormente, ou seja, todo o código inserido no Hook de efeito é lido a cada renderização e re-renderização do componente. Inclusive, esse é o seu comportamento padrão. Ainda a respeito do código, em que temos a modificação de um elemento na árvore DOM, é possível realizar muito mais com o Hook em questão, por exemplo, sendo uma abordagem muito comum na chamada de APIs, no carregamento do componente, a fim de popular o mesmo. Nesse sentido, poderíamos chamar uma API que retorne uma coleção/lista de produtos e, com esses dados, popular nossa página, mostrando os detalhes dos produtos.

Avançando um pouco em relação ao useEffect, há um ponto importante que nem sempre traz a melhor experiência: seu comportamento padrão de executar a cada renderização/re-renderização. Se em algumas situações esse comportamento é importante, em outros, não. Logo, é possível alterarmos esse comportamento padrão de duas formas: dizendo ao Hook que execute apenas uma vez, na renderização, ou que seja executado apenas quando determinado elemento for alterado. Para isso, utilizamos o array de dependências, que é um argumento opcional. Vamos a mais um exemplo de código, em que passaremos um array de dependência vazio para o useEffect, fazendo com que ele seja renderizado apenas no carregamento do componente. Utilizaremos o último código acima, modificando-o para que fique assim:

```
import React, { useState, useEffect } from 'react';
function App() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    // Atualiza o título do documento usando a API do browser
    document.title = `Você clicou ${count} vezes`;
    console.log('Componente re-renderizado');
  },[]);

  return (
    <div>
      <p>Você clicou {count} vezes</p>
      <button onClick={() => setCount(count + 1)}>
        Clique aqui
      </button>
```

Note que a única alteração foi a inclusão do array vazio, representado por [], antes do parêntese que fecha a declaração do useEffect. Teste esse código. A partir de agora, o título do documento não será mais atualizado a cada clique no botão, já que informamos que o hook deverá ser executado apenas na renderização do componente. Como já dito, podemos também modificar o comportamento default definindo que sua execução

dependerá da alteração de um ou mais valores/elementos. Nesse caso, basta incluirmos, dentro dos [] usados no exemplo anterior, os valores que deverão ser "ouvidos". Tais elementos podem ser um state ou uma props, por exemplo. Além disso, podemos passar mais de um item, separando-os por vírgula.

Atividade Extra

A documentação oficial do React tem algumas seções falando a respeito dos Hooks. Caso queiram se aprofundar no assunto, seguem alguns links:

https://pt-br.reactjs.org/docs/hooks-state.html#gatsby-focus-wrapper **(Acesso em 29/09/2022)**

https://pt-br.reactjs.org/docs/hooks-rules.html#gatsby-focus-wrapper **(Acesso em 29/09/2022)**

Referência Bibliográfica

MORGAN, Joe. **How to code in React.js.** Digital Ocean, 2021.

Atividade Prática Módulo 9



Título da Prática: Combinando os hooks useState e useEffect.

Objetivos: Permitir ao aluno verificar, na prática, seu entendimento sobre os React Hooks.

Materiais, Métodos e Ferramentas: IDE VS Code

Atividade Prática

Os React Hooks são funções que facilitam o dia a dia do desenvolvedor, disponibilizando funções que só seriam possíveis, de outra forma, com a escrita de muitas linhas de código. Entre os Hooks disponibilizados pela biblioteca, o useState e o useEffect estão entre os mais utilizados. Hoje, você deverá criar um componente fazendo uso desses dois hooks, conforme requisitos a seguir:

- Armazene no componente a temperatura ambiente, cujo valor inicial será de -5;

- Mostre o valor da temperatura no navegador;

- Crie dois botões que permitam ao usuário aumentar ou diminuir a temperatura, de 5 em 5 graus;

- O valor da temperatura também deverá ser exibido no título da página web;

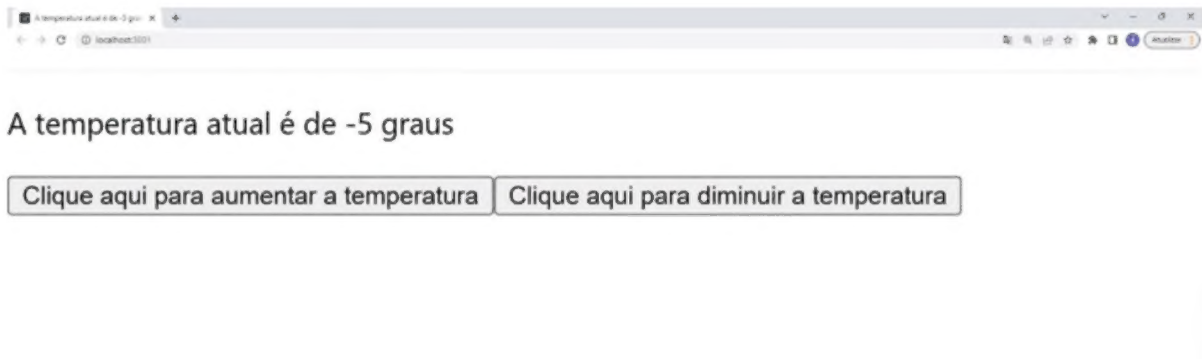
- O texto a ser exibido, tanto no corpo da página, quanto no título deverá ser: "A temperatura atual é de N graus" (onde N equivalerá

ao valor, incrementado ou decrementado, conforme interação do usuário;



Gabarito Atividade Prática

Ao final da execução dos passos descritos acima, o aluno deverá ver a seguinte página no navegador web:



Na IDE VS Code, o código deverá estar desta forma:

```javascript
import React, { useState, useEffect } from 'react';
const Atividade = () => {
  const [temperatura, setTemperatura] = useState(-5);

  useEffect(() => {
    document.title = `A temperatura atual é de  ${temperatura} graus`;
  });

  return (
    <div>
      <p>{`A temperatura atual é de  ${temperatura} graus`}</p>
      <button onClick={() => setTemperatura(temperatura + 5)}>
        Clique aqui para aumentar a temperatura
      </button>
      <button onClick={() => setTemperatura(temperatura - 5)}>
        Clique aqui para diminuir a temperatura
      </button>
    </div>
  );
}

export default Atividade;
```

Ir para exercício